9.3.5.4 Denotational and connotational subtypes (EXPRESS-2)
When an entity instance is created it identifies those entity datatypes it is an instance of, and unless the datatype information is changed by using the *extend* or *drop* built in actions it remains an instance of those datatypes. Some subtypes only add constraints to the information held for the supertype. An entity instance which is created as an instance of the supertype but which conforms to the constraints of the subtype cannot be used as the subtype. A connotational subtype allows an instance of a particular supertype to be used as if it were a subtype instance also even though the instance is not declared to be of that particular subtype. The connotational subtype construct directs an information base to treat a supertype instance as if it were a subtype instance if it meets all constraints specified in the subtype. If CONNOTATIONAL is not specified then the supertype instance may not be considered as an instance of the subtype, irrespective of any constraints specified in the subtype. In such a case the subtype is said to be denotational and the instance must be instantiated as the subtype to be an instance of the subtype.

Rules and restrictions:

a) A CONNOTATIONAL subtype shall not contain explicit attribute declarations;

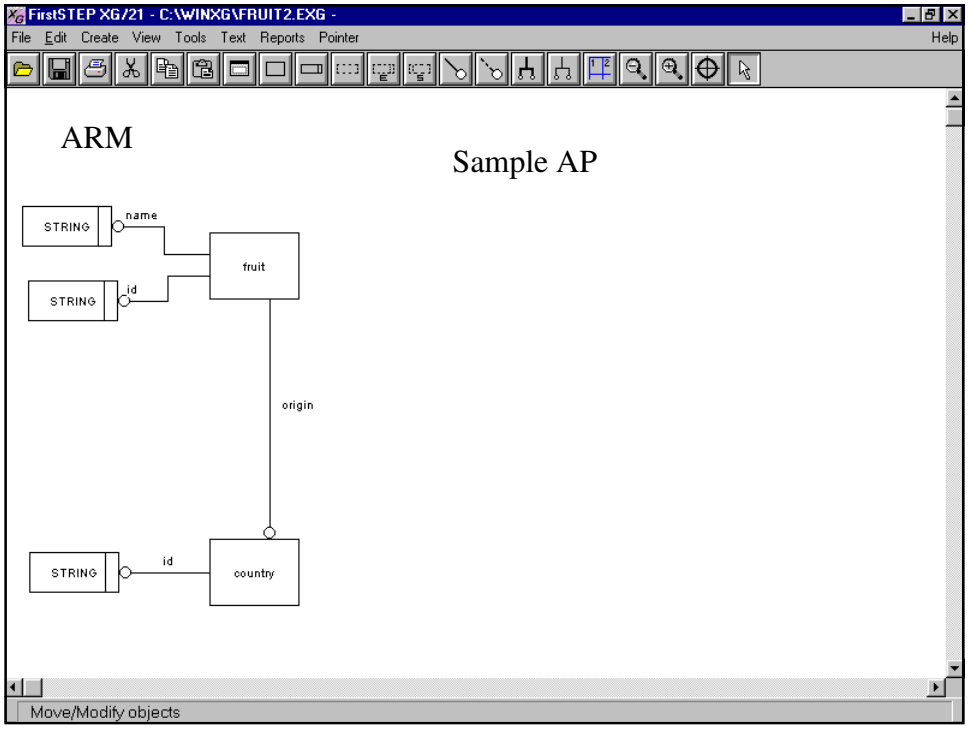
b) A CONNOTATIONAL subtype may not be constrained to be either ABSTRACT or TOTAL_OVER.

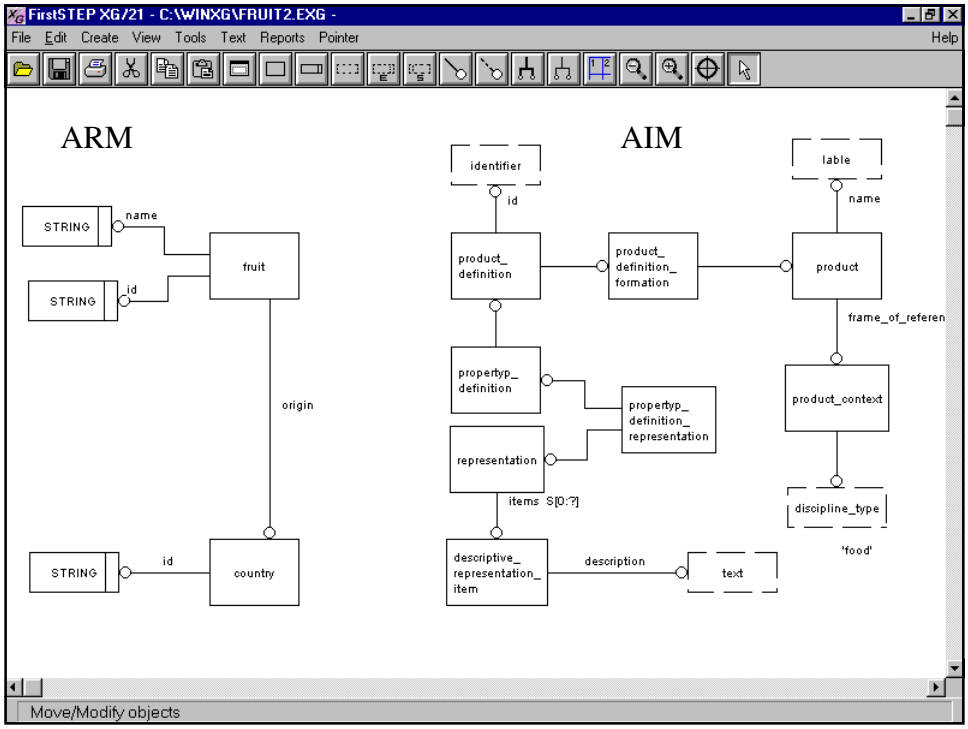
c) An instance of a supertype of a denotational subtype may not be considered to be an instance of the denotational subtype unless explicitly instantiated to be.

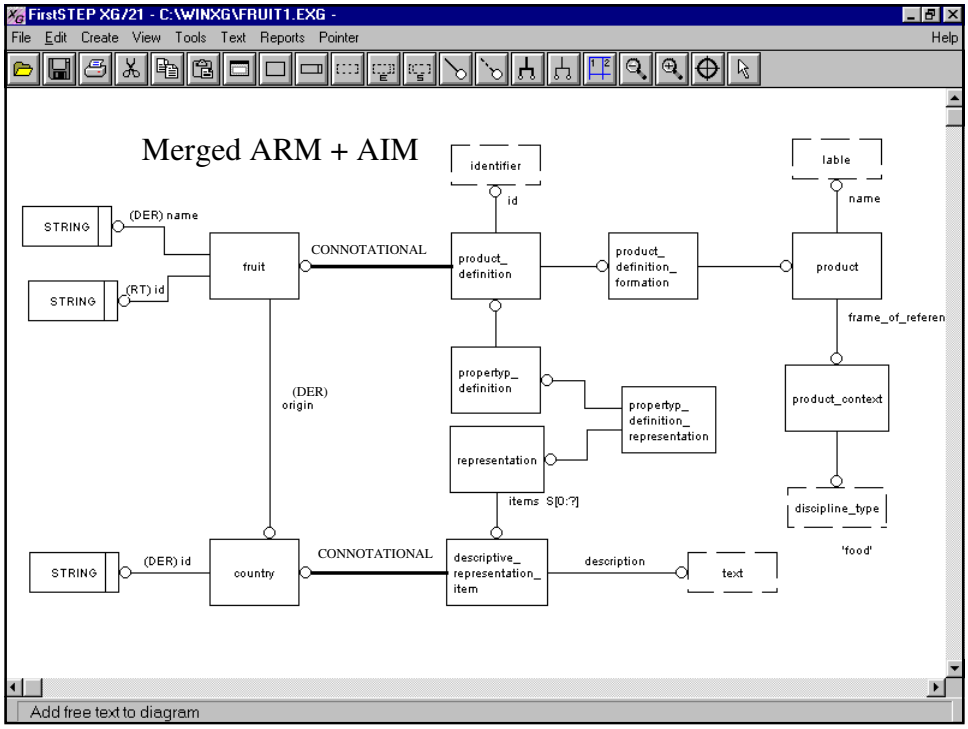
EXAMPLE In the following example two subtypes are defined, one a connotational subtype, the other a denotational subtype.

```
ENTITY person;
        name : personal_name;
        age : INTEGER;
END_ENTITY;

ENTITY male
        SUBTYPE OF (person);   ...
END_ENTITY;

ENTITY pensioner
        CONNOTATIONAL SUBTYPE OF (person);
WHERE
        old: age > 65;
END_ENTITY;

FUNCTION pension (subject : pensioner) : REAL;
        ...
END_FUNCTION;
```

```
LOCAL
  fred : male := person('Fred', 70)||male();
  (* Fred may be used as a pensioner since it meets the constraints
  of the connotational subtype pensioner *)
  bert : male := person('Bert', 65)||male();
END_LOCAL;
...
income := pension(fred);
  -- valid, Fred meets the pensioner constraints.
income := pension(bert);
  (* invalid, Bert cannot be considered a pensioner, and therefore
  cannot be passed as a parameter to the pension function,
  which returns indeterminate (?).*)

---------- p21 --------
#55=MALE('Mike',70);
#66=MALE('Jack',65);
```


FirstSTEP XG/21 - C:\WINXG\FRUIT2.EXG -
File Edit Create View Tools Text Reports Pointer Help
ARM
Sample AP
STRING
name
STRING
id
fruit
origin
STRING
id
country
Move/Modify objects


FirstSTEP XG/21 - C:\WINXG\FRUIT2.EXG -
File Edit Create View Tools Text Reports Pointer Help
ARM
AIM
STRING
name
STRING
id
fruit
origin
STRING
id
country
identifier
id
product_ definition
product_ definition_ formation
lable
name
product
frame_of_referen
propertyp_ definition
propertyp_ definition_ representation
representation
product_context
items S[0:?]
discipline_type
'food'
descriptive_ representation_ item
description
text
Move/Modify objects



Advantages of CONNOTATIONAL SUBTYPE for Modules

- a mapping table would no longer be needed
- the ARM-Express would directly be integrated in the AIM. ARM entities would become connotational subtypes of AIM entities.
- ARM attributes would become derived attributes.
- As a result ARM and AIM would be in one schema.
- a huge amount of redundancy in the mapping tables could be eliminated.
- The result (part21) would still be fully compatible with what we have today
- No identity problems like in the EXPRESS-X "views"
- Only a one page extension in the EXPRESS-Amendment is needed.

CONNOTATIONAL SUBTYPE is a challenge
take advantage of it